REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
April 2004	Technical Paper	April 2004

4. TITLE AND SUBTITLE
Ontology-Driven Translator Generator for Data Display Configurations

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Charles Jones PhD

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)
412 TWENTI
Air Force Flight Test Center (AFFTC)
Edwards AFB, CA 93524

8. PERFORMING ORGANIZATION REPORT NUMBER
PA-04141

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
412 TW/ENTI
Air Force Flight Test Center (AFFTC)
Edwards AFB, CA 93524

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
N/A

12. DISTRIBUTION / AVAILABILITY STATEMENT
A Approved for public release; distribution is unlimited.

20040921 039

13. SUPPLEMENTARY NOTES
CC: 012100 CA: Air Force Flight Test Center Edwards AFB

14. ABSTRACT

This paper presents a new approach for the effective generation of translator scripts that can be used to automate the translation of data display configurations from one vendor format to another. Our approach uses the IDEF5 ontology description method to capture the ontology of each vendor format and provides simple rules for performing mappings. In addition, the method includes the specification of mappings between a language-specific ontology and its corresponding syntax specification, that is, either an eXtensible Markup Language (XML) Schema or Document Type Description (DTD). Finally, we provide an algorithm for automatically generating eXtensible Stylesheet Language Transformation (XSLT) scripts that transform XML documents from one language to another. The method is implemented in a graphical tool called the Data Display Translator Generator (DDTG) that supports both inter-language (ontology-to-ontology) and intra-language (syntax-to-ontology) mappings and generates the XSLT scripts. The tool renders the XML Schema or DTD as trees, provides intuitive, user-friendly interfaces for performing the mappings, and provides a report of completed mappings. It also generates data type conversion code when both the source and target syntaxes are XML Schema-based. Our approach has the advantage of performing language mappings at an abstract, ontology level, and facilitates the mapping of tool ontologies to a common domain ontology (in our case, Data Display Markup Language or DDML), thereby eliminating the $O(n^2)$ mapping problem that involves a number of data formats in the same domain.

15. SUBJECT TERMS
eXtensible Markup Language (XML)
Document Type Description (DTD)
Data Display Markup Language (DDML)
eXtensible Stylesheet Language Transformation (XSLT)
Data Display Translator Generator (DDTG)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
					Charles Jones PhD.
a. REPORT	b. ABSTRACT	c. THIS PAGE	Unclassified Unlimited	9	19b. TELEPHONE NUMBER *(include area code)* 661-275-4419
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED			

ONTOLOGY-DRIVEN TRANSLATOR GENERATOR FOR DATA DISPLAY CONFIGURATIONS

Ronald Fernandes, Michael Graul, and Burak Meric
Knowledge Based Systems, Inc.
College Station, TX 77840.
{rfernandes, mgraul, bmeric}@kbsi.com

Charles H. Jones
412 TW/ENTI
Edwards AFB, CA 93524-8300
charles.jones@edwards.af.mil

ABSTRACT

This paper presents a new approach for the effective generation of translator scripts that can be used to automate the translation of data display configurations from one vendor format to another. Our approach uses the IDEF5 ontology description method to capture the ontology of each vendor format and provides simple rules for performing mappings. In addition, the method includes the specification of mappings between a language-specific ontology and its corresponding syntax specification, tThat is, either an eXtensible Markup Language (XML) Schema or Document Type Description (DTD). Finally, we provide an algorithm for automatically generating eXtensible Stylesheet Language Transformation (XSLT) scripts that transform XML documents from one language to another. The method is implemented in a graphical tool called the Data Display Translator Generator (DDTG) that supports both inter-language (ontology-to-ontology) and intra-language (syntax-to-ontology) mappings and generates the XSLT scripts. The tool renders the XML Schema or DTD as trees, provides intuitive, user-friendly interfaces for performing the mappings, and provides a report of completed mappings. It also generates data type conversion code when both the source and target syntaxes are XML Schema-based. Our approach has the advantage of performing language mappings at an abstract, ontology level, and facilitates the mapping of tool ontologies to a common domain ontology (in our case, Data Display Markup Language or DDML), thereby eliminating the $O(n^2)$ mapping problem that involves a number of data formats in the same domain.

KEYWORDS

Data Display Configuration, T&E Environment, Neutral Format, XML, XSLT, Automated Translator Generator

INTRODUCTION

In a Test & Evaluation (T&E) environment, data display systems play a critical role and need to be quickly assembled, programmed, and tested for both real-time and post-test analysis. Data displays have a wide range of parameters, attributes, dynamics, and data sources. The T&E systems, at different locations, need to transfer and share telemetry data among themselves. In the traditional approach, each of these systems would apply its unique, and quite complex, display setups. To compound this situation, there are a variety of data display vendors, each requiring its own data display specification. As a consequence, the time required to set up and check these data displays is significant. Most often, the only way to transfer data displays between display applications is to manually recreate the displays in another environment. Also, a change in one of the displays requires manual changes in the other related displays. Thus, the absence of automated translators and corresponding requirement for manual setup cause significant delays in test programs.

Even if re-creation of data displays were to use a set of automated translators, the absence of a neutral format would require a total of $n(n\text{-}1)$ translators to be built, where n is the number of data display systems. To mitigate this problem, DDML was developed as an XML-based neutral format for data displays [1]. As a result, the number of translators that need to be developed reduces to two unidirectional translators between the neutral format and each vendor-specific format, for a total of $2n$ translators. An additional advantage of having DDML is that a change in one of the vendor formats requires the recoding of only the translators between that format and the neutral format. Figure 1~~Figure 1~~ illustrates the benefit of using DDML.

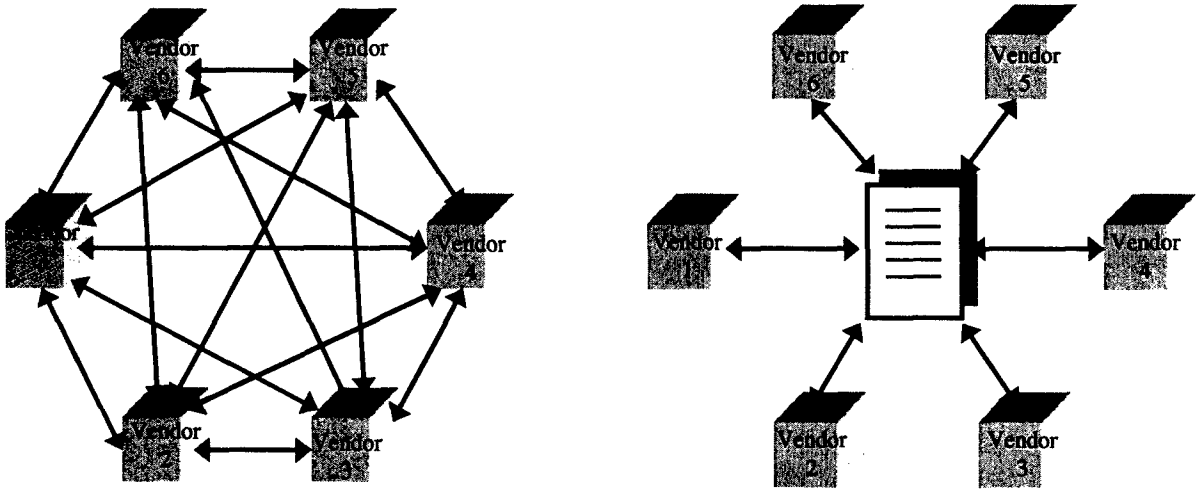

Figure 1: Code Development Effort for Translators Without and With Uuse of a Neutral Format

We can even do better by ***automatically generating*** the translator code, which can then be compiled, tested, and deployed. As a result, the focus of the configuration translation effort is on the modeling of the data display specification and not on the development of translator code. In that sense, it will be similar to using Computer-Assisted Software Engineering (CASE) tools to develop object-oriented software models (say, in the Unified Modeling Language [UML]) and to automatically generate the target code for compiling. A great advantage in this approach is that a change in one of the vendor formats would not require programming of any sort – the model would simply have to be

changed and the translator code would be regenerated to reflect the changes. As shown in Figure 2~~Figure 2~~, the programming effort to develop translators for a T&E system of *n* applications would be non-existent; the only effort needed would be to perform semantic and syntactic modeling for each vendor format in the framework.

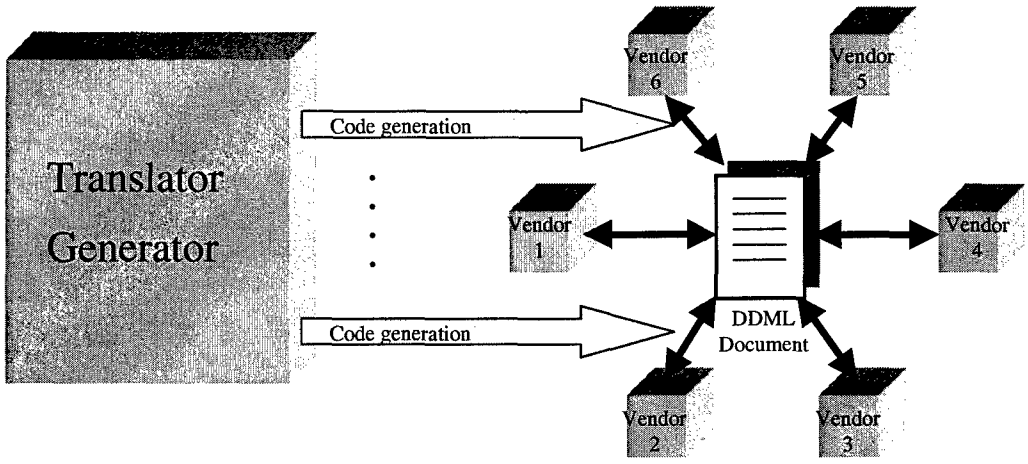

Figure 2: Automated Code Generation for Translators to and from Neutral Format

In this paper, we describe an ontology-driven methodology for managing syntactic mappings between DDML and any other data display language. An ontology-driven approach is crucial because ontology not only captures the vocabulary, but also provides a rich description of the meanings of the terms in that vocabulary in the form of a set of logical axioms. An ontology of display systems, for example, will provide a list of concepts that are needed to build a data display application (e.g., display model, display objects, slider, data variables, etc.) together with a formal description of these concepts, a specification of their properties, and the identification of the relationships that they bear to one another.

Building an ontology is a non-trivial operation that requires expertise in both the domain being captured and formal specifications. In the case of information translation, however, the ontologies to be built are formal specifications of the meta-models of the chosen applications. One such formal specification for ontologies is the IDEF5 ontology language [2], a theoretically and empirically well-grounded language specifically designed to assist in creating, modifying, and maintaining ontologies both graphically and using a first-order textual language.

Our approach for data display format mapping to DDML is to first capture the ontology of each language using IDEF5 and then to provide both the syntax-to-ontology mappings within each language and the ontology-to-ontology mappings between the two languages. There are numerous advantages to this approach. First, the method serves as a convenient manner for capturing, storing and disseminating mapping knowledge relating to two tools or two related domains. Second, it facilitates the automatic generation of translation scripts between two languages. Finally, because mapping takes place at the semantic level, translation scripts can be regenerated whenever the syntactic format of either language changes. Our current implementation of the tool that supports the methodology includes display formats that can be persistently stored in XML.

THE IDEF5 ONTOLOGY LANGUAGE

Our methodology is based on the IDEF5 ~~O~~ontology description capture language [1] for building and representing ontologies. IDEF5 provides a theoretically and empirically well-grounded method specifically designed to assist in creating, modifying, and maintaining ontologies. Standardized procedures, the ability to represent ontology information in an intuitive and natural form, and higher quality results enabled through the use of IDEF5 also serve to reduce the cost of these activities.

Supporting the ontology development process are IDEF5's ontology languages. There are two such languages: the IDEF5 S~~s~~chematic ~~l~~Language and the IDEF5 ~~e~~Elaboration ~~l~~Language. The S~~s~~chematic ~~l~~Language is a ~~g~~Graphical ~~l~~Language specifically tailored to enable domain experts to express the most common forms of ontological information. This ~~g~~Graphical ~~l~~Language enables average users both to input the basic information needed for a first-cut ontology and to augment or revise existing ontologies with new information. The other language is the IDEF5 ~~e~~Elaboration ~~l~~Language, a structured textual language that allows detailed characterization of the elements in the ontology.

Various diagram types, or schematics, can be constructed in the IDEF5 Schematic Language. The purpose of these schematics, like that of any representation, is to represent information visually. Thus, semantic rules must be provided for interpreting every possible schematic. These rules are provided by outlining the rules for interpreting the most basic constructs of the language and applying them recursively to more complex constructs.

However, the character of the semantics for the Schematic Language differs from the character of the semantics for other graphical languages. Specifically, each basic schematic is provided only with a default semantics that can be overridden in the Elaboration Language. The reason for this is that the chief purpose of the Schematic Language is to serve as an aid for the construction of ontologies – they are not the primary representational medium for storing them. That task falls to the Elaboration Language. The Schematic Language is, however, useful for constructing first-cut ontologies in which the central concern is to record, in a rough way, the basic elements that exist in a domain, their characteristic properties, and the salient relations that can be obtained among objects of those kinds and among the kinds themselves. Consequently, the basic constructs of the Schematic Language are designed specifically to capture this type of information.

ONTOLOGY-BASED DATA DISPLAY MAPPINGS

Our approach for XML schema mappings is to first capture the ontology of each language using IDEF5 and then to provide both the syntax-to-ontology mappings within each language and the ontology-to-ontology mappings between the two languages. An example of such mapping is provided in Figure 3~~Figure 3~~, which shows two data display formats mapped to DDML.

We have implemented the mapping methodology in a software tool called Ontology-Driven Translator Generator (ODTG). A typical use-case of ODTG is as follows:~~:~~

1) The user specifies the source schema or DTD, which is parsed and displayed as a tree.
2) The user specifies or develops the ontology of the source language.
3) The user maps the source schema (DTD) to the source ontology.
4) The user maps the source ontology to the pre-built ontology of the target XML.
5) The tool generates the XSL-based translator script that contains the mapping rules.
6) The user modifies the XSL mapping rules, if necessary.
7) The user uses an XSLT processor to convert a source XML to target XML.

Figure 4~~Figure 4~~ shows the XSLT script generation process and its use by an XSLT engine.

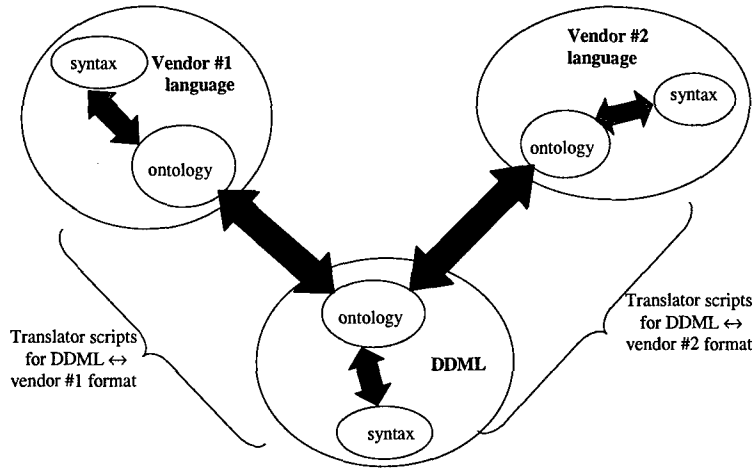

Figure 3: Vendor #1 ↔ DDML ↔ Vendor #2 Translator Generation Scripts

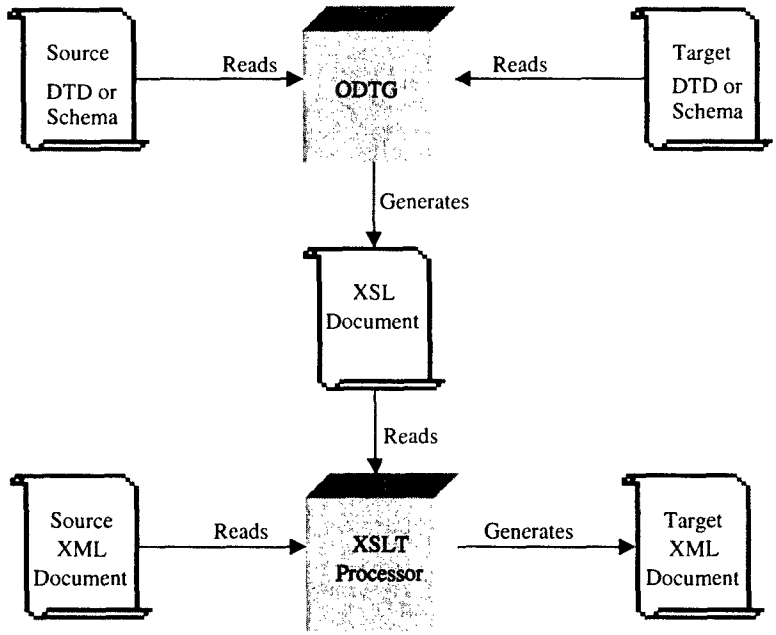

Figure 4: XSLT Generation Process and its Uuse in XML Document Conversion

XSLT GENERATION

The generation of XSLT [3] that encodes the mapping rules requires the end-to-end mapping of the source-syntax ↔ source-ontology ↔ target-ontology ↔ target-syntax for every node of the syntax trees in order to generate a complete rule-set. The XSLT generation outline is described below.

Perform a ~~D~~depth-~~F~~first ~~S~~search on the target tree

For each node of the target tree, find the corresponding node in the source tree

If a corresponding node is not found

If the node is not a leaf node then generate XSLT tag statements and continue traversing

Else if a corresponding node is found

If the corresponding node is a leaf node then generate XSLT value statements

Else if the corresponding node is not a leaf node then

Generate XSLT template statements

Generate attribute/element XSLT statements according to the type of the node

Continue traversing

The algorithm processes each node of the target tree by traversing it with depth-first search (DFS). As the traversal method of the XSLT processors, DFS is the natural choice when dealing with XSLT documents. When a node is processed with DFS, its corresponding node in the source tree is found by examining the source-syntax ↔ source-ontology ↔ target–ontology ↔ target-syntax mappings. The node is then processed according to its type (attribute vs. element), location in the tree, and the location and type of the corresponding node.

An important issue with the XSLT generation algorithm is how to determine the target node that corresponds to a node in the source document. Currently, the algorithm assumes that there is a one-to-one relationship between the source nodes and the target nodes. This restriction will be overcome in the future.

IMPLEMENTATION

The ODTG tool is implemented in Java. A user can create new ontologies, syntaxes, and mappings by using the tool. It is also possible to load existing mappings, syntaxes, and ontologies saved as a persistent XML project file. Thus the tool allows incremental development of various languages in a domain. The tool has three main window types.

1. Language Window: This window enables the user to specify the syntax, ontology, and syntax ↔ ontology mapping. It consists of two panels: t~~T~~he left panel shows the syntax tree, and the right panel shows the IDEF5-based ontology graphical view. A mapping between the syntax and ontology is shown by highlighting the corresponding mapped elements or objects when a particular element of the language is highlighted. The language window is shown in Figure 5~~Figure 5~~.
2. Ontology Map Window: This window enables the user to specify the ontology ↔ ontology mappings between two languages. The ontology-ontology map window for DDML and IADS [4] languages is shown in Figure 6~~Figure 6~~.
3. Mapping List Window: This is a summary mapping of window types (1) and (2) above. It consists of two panels, both of which are tree views. The mapping list window for DDML is shown in Figure 7~~Figure 7~~.

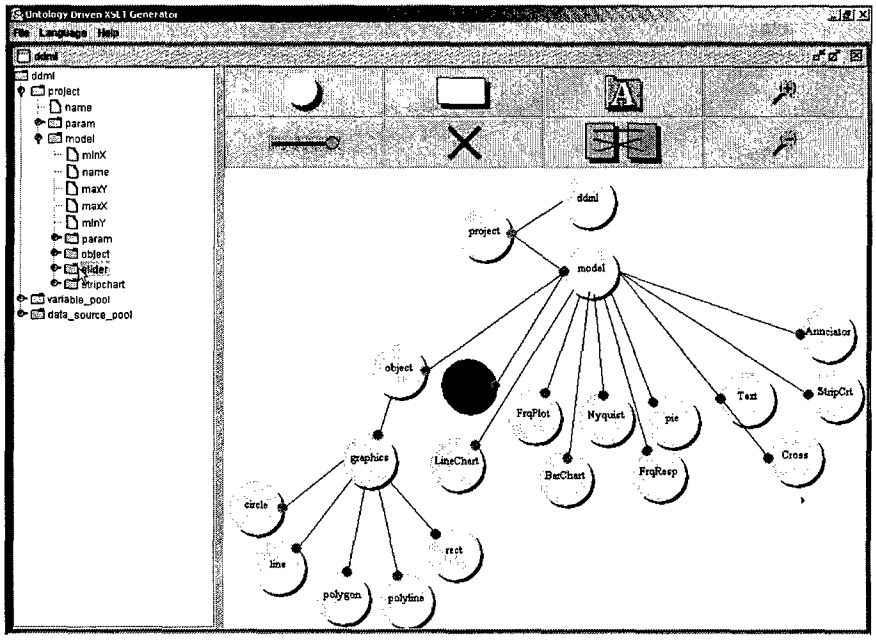

Figure 5: Language Window Showing the DDML Language

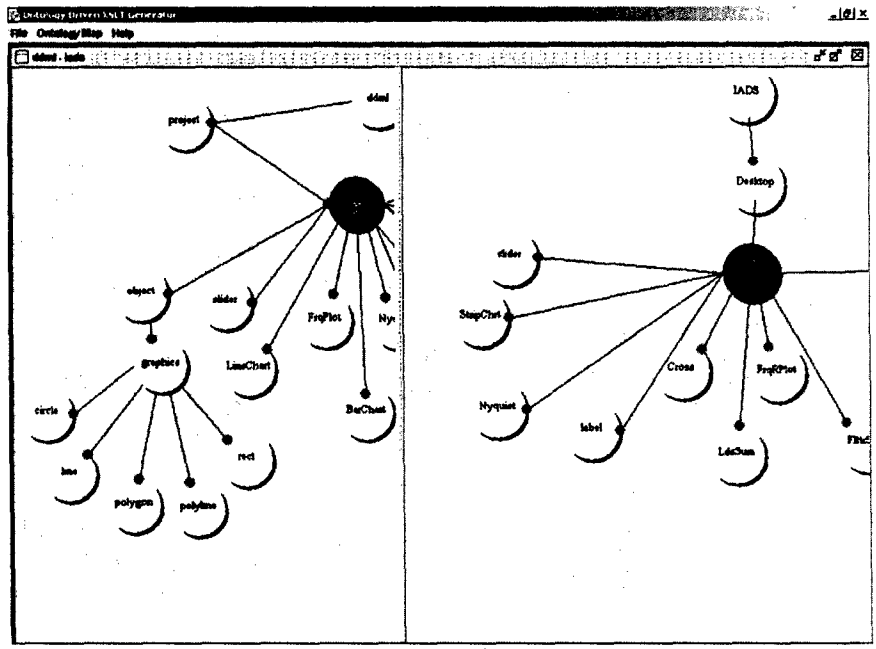

Figure 6: Ontology Map Window for DDML and IADS

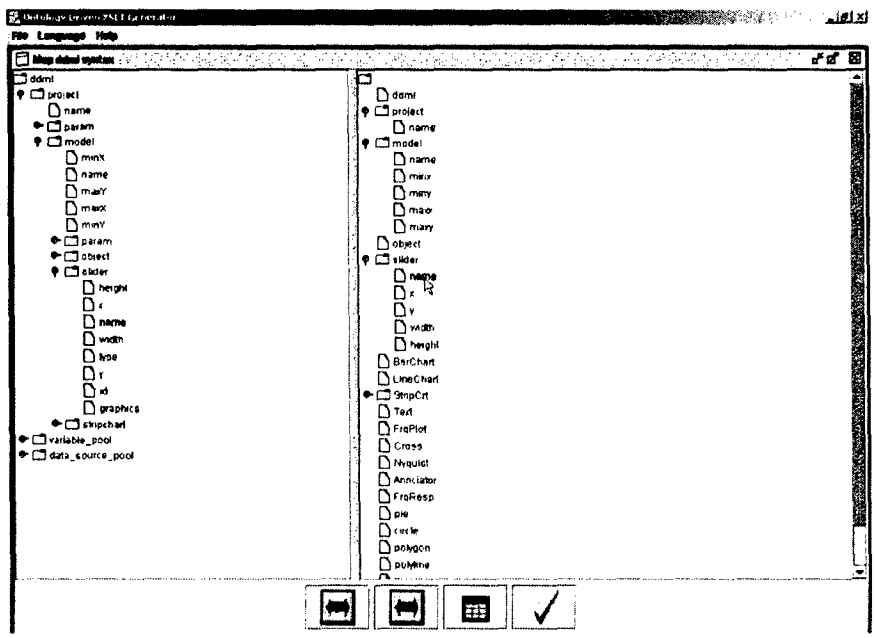

Figure 7: DDML Syntax ↔ Ontology Map Window

The syntax reader module of the Language Window supports both DTDs and XML schemas. Once the ontologies and the mappings are complete, the XSLT files can be generated automatically. In our test case, we XML-ized the IADS data display format to generate XSLT scripts. The generated XSLT file for DDML to the XML-ized IADS was validated by executing the script using freely-available XSLT processing engines (see Figure 4~~Figure 4~~) such as Xalan [5].

Current work in progress includes generating translators for non-XML-based display formats.

CONCLUSIONS

In this paper, we have described an intuitive approach for effective generation of translator scripts that can be used to automate the translation of data display configurations from one display vendor format to another. Our approach uses the IDEF5 ontology description method to capture the ontology of each vendor format and makes use of the Data Display Markup Language (DDML) as a neutral format among various vendor formats. In addition, the method includes the specification of mappings between the vendor's tool ontology and its corresponding syntax specification (XML Schema or DTD). Finally, the method includes the use of an algorithm for automatically generating XSLT scripts that transform XML documents between DDML and XML-based vendor formats.

The method is implemented in a graphical tool called the Data Display Translator Generator (DDTG) that supports both the inter-language (ontology-to-ontology) and intra-language (syntax-to-ontology) mappings and generates the XSLT scripts. The tool renders the XML Schema or DTD as trees, provides intuitive user-friendly interfaces for performing the mappings, and provides a report of completed mappings. Our approach has the advantage of performing language mappings at an abstract, ontological level, and facilitates the mapping of data display tool ontologies to DDML, thereby eliminating the $O(n^2)$ mapping problem involving a number of display formats. An important benefit of ODTG is a 'no programming required' user interface that can enable modelers and domain experts to interchange models without having to know details of XSLT. Because mappings take place at the semantic level, translation scripts can be regenerated whenever the ontology or syntax of either language changes.

REFERENCES

[1] Burak Meric, Michael Graul, Ronald Fernandes, and Charles H. Jones, "Design of an interlingua for data display systems," ITC 2003,

[2] The IDEF5 Ontology Capture Method, http://www.idef .com/idef5.html

[3] XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt

[4] Symvionics' Interactive Analysis and Display System (IADS™), http://www.symvionics.com/products/IADS.html

[5] XALAN, Apache's XSLT Transformation Engine, http://xml.apache.org/xalan-j/index.html